Using a ILP algorithm to learn Logic Programs for Reasoning about Actions

David Lorenzo[1] and Ramon P. Otero[2]

[1] Computer Science Dept., Univ. Corunna (Spain)
david@dc.fi.udc.es
[2] Computer Science Dept., Univ. of Texas at El Paso (USA)
otero@cs.utep.edu

Abstract. ILP is suitable to learn action descriptions of dynamic domains. We introduce an algorithm, inspired on Progol, but adapted to cope with the particular issues of action descriptions. The algorithm is shown in an application in Cognitive Robotics.

1 Introduction

We focus on learning descriptions of dynamic domains that can be characterized by logic-based formalisms of action. These formalisms aim to be a theory of dynamics, grounded on a strong mathematical foundation, where system's behaviors are viewed as appropriate logical consequences of the domain's description [8]. Recently there has been much progress in formulating theories of action, particularly in progressing from simple and/or restricted theories and "example centered approaches", to general and provenly correct theories that incrementally consider various specification aspects such as: concurrent actions, etc. This allows to consider increasingly more complex tasks, e.g., Cognitive Robotics [10].

Some of these formalisms have a Logic Programming (LP) counterpart what makes it feasible to study the integration with Inductive Logic Programming (ILP). This work deals with a logical approach to modeling dynamical systems based on a dialect of first order logic called the Situation Calculus (SC). As to learning, the apprentice is provided with a sequence of states and it must infer how properties of a domain are (directly/indirectly) affected by the execution of actions, or otherwise are subject to the general law of persistence.

There are many approaches to learning dynamic systems using induction methods, among other [4, 11, 3]. Unlike [11], we consider the learning process and the exploratory process as two separate processes, such that we restrict to a finite dataset. Unlike [4], that presents an approach to robot learning applied to physical robots at the lowest level of control, the assumptions we have made are often appropriate for the higher level of control of physical robots. For instance, action $go(forward)$ in the office scenario actually means translating forward with a certain speed and at the same time avoiding obstacles and avoiding a crash. Despite this, it makes sense to model system identification in a variety of interesting tasks and environments in terms of inferring action theories The most distinguishing aspect of action theories is that the specification of actions and their effects shall be as intuitive and natural as possible. Furthermore, the use of a logical formalism allows, e.g., to make plans that will take us to particular kind of worlds and explain observations about the state of the world in terms of what actions might have taken place, or what state the world might have been before, or both.

This paper is organized as follows. Sections 2 and 3 show how classical action theories can be learned by adapting current *ILP* methods for static domains. The paper concludes with some discussion and future directions for research.

2 Learning Logic Programs for Reasoning about actions

A formal framework to reason about actions and change requires the basic notion of a situation, which is a partial snapshot of the world at a particular instant of time. A *domain description* consists of two nonempty sets: a set $\mathcal{F}$ of fluent names, and a set $\mathcal{A}$ of action names. An action, if executed in some state, leads to a "resulting" state. So-called fluents serve to describe situations and are properties whose truth values may change in the course of time.

SC theories can be represented as logic programs with a fixed clausal structure where any aspect of a system which can change as the result of an action is indexed by a situation term. A situation argument may be in the form of a constant s_i or else a situation resulting after executing an action $do(act, s)$, and a special situation s_0 is included to represent the initial situation. A sequence of situations is encoded in the form of chains `do(act,do(act,..do(act,s0)..))` starting from the initial situation.

Formally we have

Definition 1 (Situation Calculus Program) A SC program is a set of clauses in the form [9]:

$$\{\neg\}Holds(f, s_0) \tag{1}$$

$$\{\neg\}Holds(f, do(a, s)) \leftarrow \pi \tag{2}$$

$$Affects(a, f, s) \leftarrow \pi \tag{3}$$

$$Holds(f, do(a, s)) \leftarrow Holds(f, s) \wedge \text{not } Affects(a, f, s) \tag{4}$$

$$\neg Holds(f, do(a, s)) \leftarrow \neg Holds(f, s) \wedge \text{not } Affects(a, f, s) \tag{5}$$

where s_0 denotes the initial situation, Formula (2 and 3) state that, in any situation, if the precondition holds then the effect will hold in the resulting situation. These axioms are called *effect axioms* or *action laws*. π does not mention the *Affects* predicate and every occurrence of the *Holds* predicate in π is of the form $Holds(F', s)$. The universal frame axioms (4) and (5) describe how the world stays the same (as opposed to how it changes).

□

Example 1 (Office scenario).

Let us consider a mobile robot navigating the office floor in Fig. 1 [1]. This example concerns the representation of knowledge about the objects in a robot's environment, and how such knowledge is acquired. We assume the state space has been reduced to a manageable size by careful choice of perception and action primitives. The actions used in this scenario are of two types: turning actions, that change the heading of the robot and navigational actions, that take the robot from one kind of node to an adjacent one. For instance, action $go(forward)$ takes the robot forward a certain distance until there is change in its sensor values regarding what is on its right. The robot's sensing mechanism can tell where the robot is located at, in terms of being next to rooms 301 to 349 or next to the elevator, as well as to detect what it is on its right. The doors of the outer rooms are painted white and the doors of the inner rooms are painted black We additionally assume that actions are duration less and have instantaneous effects.

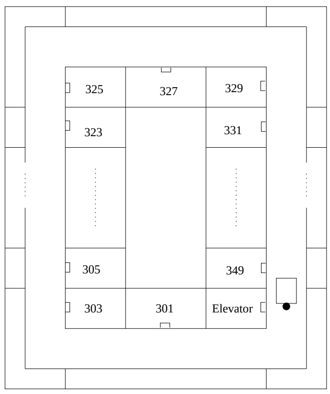

Fig. 1. Mobile robot in a simple office floor

We will focus on a sub-scenario that uniquely contains the navigational and turning actions found in Table 1. Actions and sensor measurements accumulated during robot routes (in a simulation), are the input to learning. Adjacent rooms are represented by means of an additional predicate `rightof(X,Y)` in the background.

The basic problem is to identify the effects of actions, values of fluents (observations) and relationships between them, describing how they progress a domain where they are performed. Examples are observations of the shape $[\neg]holds(f, do(a, s))$, representing the caused truth values for fluents at each situation, whereas negative examples are negations of observations. The use of classical negation is needed for reasoning about actions because an agent needs to distinguish between successful actions, unsuccessful actions, and actions with

Fluents	Description
on_rt/1	information regarding what it's on its right
app_corner/0	the robot is approaching a corner
at_room/1	the current room number where the robot lies
at_elevator/0	the robot is in front of the elevator
Actions	Description
go/1	takes the robot forward/backward a certain distance
turn_rt90/0	first makes the robot turn 90 degrees to the right and then makes it go forward until the wall is on its right.
turn_180/0	makes the robot turn 180 degrees

Table 1. Fluents and actions for the office scenario

an unknown outcome. In the current implementation, negation is reified and a consistency control added, so that negative values of fluents can be dealt with.

```
holds(at_elevator,-,s0).
holds(on_rt(white_door),+,s0).
holds(at_room(305),+,do(go(forward),s0)).
holds(on_rt(black_door),+,do(turn_180,s0)).
    ...
affects(go(forward),at_room(305),s0).
affects(turn_180,on_rt(black_door),s0).
    ...
```

With the universal inertia axiom (eq. 4,5) included in the BK *during* learning, positive examples are only explicitly given for those situations where a fluent changes, whereas the inertia axiom propagates not affected truth values from one situation to the next one, completing every situation. The inertia axiom requires an additional predicate `affects/3` to be added to BK when the value of a fluent changes. □

Learning action theories is basically a multiple predicate learning problem (MPL)[7], where fluent definitions can be possibly interdependent. Progol is basically a single-predicate learning *spl* algorithm, however, it is able to learn multiple predicates in the order of head mode declarations. Despite this, it does not manage most common derived problems, thus, it is not possible to solve a *mpl* problem by putting together the definitions learned for each fluent f_i.

When learning a fluent f_i, examples for f_j with $j \neq i$ are used as additional BK together with any rules previously induced. However, the induced theory may result globally inconsistent when an overgeneral clause entails facts not in the evidence. We have considered two basic cases: in the first one we assume that narratives are complete -not that all possible examples are provided, but that, no fluent has missing values in the situations included in the training set. More formally we have that, for each situation s included in the training set, fluents have no missing values in any situation between the initial situation s_0 and s.

This is achieved by making a kind of CWA over inertia values, such that once we supply values for all fluents in the initial situation, we specify values for all possible situations.

In the second case, presence of global inconsistency must be checked after each new rule. Basically, for theories without cycles, a simple change in the order at which fluents are learned -following the dependences chain- the theory will be globally consistent. In general, this approach suffers from the order dependency problem, and a backtracking mechanism should be used or additional methods [2].

3 A prototype

A prototype consisting of a top-down ILP algorithm has been integrated and implemented with the Yap Prolog interpreter. The design methodology is to present the user with a Prolog interpreter augmented with inductive capabilities. The implemented ILP algorithm is mostly based on Progol [6]. Built-in predicate `domain/1` loads the training set including positive examples, (possibly) negative examples, and domain-dependent background including arbitrary logic programs. BK implicitly includes formulas (1,4 and 5) whereas formulas (2 and 3) will be the result of learning. Clauses for `affects/3` are duplicates of those of `holds/3`. Background knowledge includes user-defined type declarations for both fluents and actions, such that they can be accessed during hypothesis formation. When no negative examples are provided, these are automatically generated but restricted to the training set –negative examples are given when the opposite value holds (persists or is caused). This results practical due to the large number of negative examples that are possible.

Before learning, a declarative bias is specified through built-in predicate `bias/2` that establishes which predicate will be generalized, and the literals allowed to appear in the body of clauses using basically mode declarations. When objects other than situations are to be considered, we need to decide generalization to be done over multiple sorts. In learning `at_room/1`, for instance, we can generalize the room number and still allow references to particular rooms, and so on.

```
bias(holds(at_room(+),#,do(#,+)),
     [holds(at_room(+),#,+),holds(on_rt(#),#,+),...]).
```

A different learning task is posed for the positive and negative values of fluents and separately for every action that is shown to cause the fluent to change in any situation. This forces the explicit representation of all the effects of an action as direct effects, and may produce unnecessarily complex clauses. The use of indirect effects is a key point for learning because some features are better predicted when they are considered *indirect effects* [5]. Such indirect effects are usually represented as consequences of general laws (constraints) describing dependences between components of the world description. However, forcing indirect effects in some fluents may produce odd generalizations or even no compression at all.

In general, some fluents can be direct effects of some actions and indirect effects of others. A simple strategy is to induce direct and indirect effects separately, take all induced clauses, and search subsets of clauses that are complete. Unfortunately, a bad selection of the first clauses can cause a "snowballing" effect over subsequent clauses in the cover. To actually find a satisfactory set of clauses, we allow the inductor to determine at each step whether a fluent should be induced as a direct or an indirect effect, thus inducing possibly a mix of axioms. Without cycles or concurrent changes, the final theory does not require a particular ordering of clauses. Pure direct effects -i.e., situations where a single fluent changes- are not considered when computing coverage of constraints, which might produce a overestimated coverage (Fig. 3).

1. If $E = \varnothing$ return B
2. Let e be the first example in E
3. Construct clause $\perp_d$ for e as a direct effect
4. Induce clause H_d from $\perp_d$
5. If e is not a pure direct effect then
   Construct clause $\perp_i$ for e as an indirect effect
   Induce clause H_i from $\perp_i$
6. Let H be the best of H_i and H_d
7. Let $B = B \cup H$
8. Let $E' = \{e : e \in E \text{ and } B \models e\}$
9. Let $E = E - E'$
10. Goto 1

Fig. 2. Progol-based covering algorithm

So-called direct effects, are indeed recursive clauses where an ordering of the situation constants is imposed. With the addition of indirect effects, fluents can refer to each other producing a non-finite recursion. Even if termination is granted (parameter h), the inability to infer any facts manifests when examples made redundant by the new clause are removed from E. A mutually recursive definition can be theoretically correct -e.g., a double implication- and produce a high compression locally, but is useless from a practical point of view[1]. Cyclic definitions can appear even in the description of perfectly well-behaved physical systems.

In well-formed action theories, every fluent must be a direct effect or an indirect effect of other fluent that eventually is a direct effect. A valid theory according to this criterion is still not valid in the basic LP semantics [5], hence, ILP would benefit of the use of a more powerful semantics. In the current implementation, coverage testing of candidate hypothesis is done by previously adding

[1] This would be perfectly valid in other semantics like stable models.

holds(f(x), v, do(a(x),s)) :- holds(g(x), v, s), ...

Fig. 3. Recursive clauses for direct effects

still non covered E_{f_i} to the BK and presence of cycles is checked after each new rule. When a cycle is found, a clause is removed and additional clauses are to be generated. Biases have to be added to ensure termination. A naive but sure way to do it is for instance, considering only direct effects. With regard to global inconsistency, when the theory includes mutual recursion no simple reordering can be done. More sophisticated methods are based on heuristic measures, where the inductor decides the ordering –similarly to the case of direct/indirect effects [7]. Currently, when this happens, learning can be done only when evidence is complete.

Example 1 (cont.)
Let us consider the office's scenario. We assume an initial state where the robot is close to room 301 with a black door on its right. Examples were taken from sensor measurements accumulated during robot routes (in a simulation) where the robot advanced in a clockwise manner. No special care is taken with noise whereas for unknown values inertia is applied (by default) so that the value used is that of the most recent situation where the fluent was known. Under these conditions, the system induced the following clauses, among others:

```
holds(on_rt(black_door),+,do(go(forward),A)) :-
      holds(on_rt(wall),+,A), holds(app_corner,-,A).
holds(on_rt(black_door),-,do(go(forward),A)) :-
      holds(on_rt(black_door),+,A).
holds(on_rt(corridor),+,do(go(forward),A)) :-
      holds(on_rt(wall),+,A), holds(app_corner,+,A).
holds(on_rt(corridor),-,do(turn_rt90,A)) :- holds(on_rt(corridor),+,A).
holds(on_rt(wall),+,do(go(forward),A)) :- holds(on_rt(black_door),+,A).
holds(on_rt(wall),+,do(turn_rt90,A)) :- holds(on_rt(corridor),+,A).
holds(on_rt(wall),-,do(go(forward),A)) :- holds(on_rt(wall),+,A).

holds(app_corner,+,do(go(forward),A)) :-
             holds(on_rt(black_door),+,A), holds(at_room(301),+,A).
holds(app_corner,+,do(go(forward),A)) :-
             holds(on_rt(black_door),+,A), holds(at_room(325),+,A).
holds(app_corner,+,do(go(forward),A)) :-
             holds(on_rt(black_door),+,A), holds(at_room(327),+,A).
holds(app_corner,+,do(go(forward),A)) :-
             holds(on_rt(black_door),+,A), holds(at_elevator,+,A).
holds(app_corner,-,A) :- holds(on_rt(corridor),+,A).

holds(at_elevator,+,do(go(forward),A)) :- holds(on_rt(wall),+,A),
             holds(app_corner,-,A), holds(at_room(349),+,A).
```

```
holds(at_elevator,-,A) :- holds(at_room(B),+,A).
```

For instance, according to the rules, what the robot has on its right changes as it advances, alternating between a wall (separating rooms) a black door and the corridor (in corners). To predict that the robot is not in the elevator, it is enough that the robot is in any room, rather than through all the actions that could take the robot far from the elevator, thus making the learning process independent of the *conductance* of the domain.

With respect to fluent `at_room/1`, generalization over rooms was not always possible. To deal with non-compressed cases, we allow that multiple `bias/2` clauses can be given where different biases are tried in the order they are provided. In this case, a second bias is given that allows to induce clauses for particular rooms, e.g., with the room next to the elevator.

```
holds(at_room(A),+,do(go(forward),B)) :-
            rightof(A,C), holds(at_room(C),+,B),
            holds(app_corner,-,B), holds(on_rt(wall),+,B),
            holds(at_room(349),-,B).
holds(at_room(A),-,do(go(forward),B)) :- holds(on_rt(wall),+,B),
            holds(app_corner,-,B), holds(at_room(A),+,B).
holds(at_room(301),+,do(go(forward),A)) :- holds(on_rt(wall),+,A),
          holds(app_corner,-,A), holds(at_elevator,+,A).
```

The results of learning can verify the correctness of control modules [1] for a robot, or more interestingly, they can be used to automatically construct control modules given a set of states and a goal. For instance, in the module to pass by a corner, when the robot approaches a corner, it must wait until the corridor is on its right and then turn 90 degrees to advance. □

4 Conclusions

A Progol-based algorithm is adapted using Logic Programming for dynamic systems, for the task of learning action theories. Future work includes among others the following topics:

- Use domains with a larger number of fluents and actions and with different noise and uncertainty levels, to show the scalability of the approach.
- Use a stable models semantics that will substitute the standard LP proof procedure to test the coverage of the examples.
- We are currently considering actions that may happen independently of the robot and which are beyond its control (exogenous), apart from those that the robot can perform, e.g., when several robots are acting in the domain.
- We are also extending the system to cope for the concurrent execution of both agent-initiated and exogenous actions through so-called *compound actions*.

Acknowledgements This research was supported in part by the Government of Galicia (Spain) grant XUGA10501B/98 and in part by the Government of Spain, grant PB97-0228.

References

1. C. Baral and S.C. Tran. Relating theories of action and reactive control. *Linkoping Electronic Articles in Computer Science*, 3(9), 1998.
2. A. Esposito, D. Malerba, and F. A. Lisi. Learning recursive theories with atre,. In *H. Prade (Ed.), Proceedings of the 13th European Conference on Artificial Intelligence*, pages 435–439. John Wiley & Sons, Chichester, England, 1998.
3. D. Hume and C. Sammut. Applying inductive logic programming in reactive environments. In S. Muggleton, editor, *ILP91*, pages 279–290, 1991.
4. V. Klingspor, K. Morik, and A. Rieger. Learning concepts from sensor data of a mobile robot. *Machine Learning*, 23:305–332, 1996.
5. D. Lorenzo and R.P. Otero. Learning action theories as logic programs. In *International Joint Conference on Declarative Programming*, pages 383–396, 1999.
6. S. Muggleton. Inverse entailment and progol. *New Generation Computing Journal*, (13):245–286, 1995.
7. L. Raedt and N. Lavrac. Multiple predicate learning in two ILP settings. *J. of the IGPL*, 4(2):227–254, 1996.
8. R. Reiter. Knowledge in action: logical foundation for describing and implementing dynamical systems, 1998. Manuscript.
9. M. Shanahan. *Solving the Frame Problem. A Mathematical Investigation of the Common Sense Law of Inertia.* The MIT Press, 1997.
10. M. Shanahan. A logical account of the common sense informatic situation for a mobile robot. *Linkoping Electronic Articles in Computer Science*, 1999. to appear.
11. W. M. Shen. *Autonomous Learning from the Environment.* Computer Science Press, 1994.